

# Segundo Trabalho de Técnicas Alternativas de Programação (Prof. Alexandre Direne - 2010/1)

## Atenção:

Este trabalho é obrigatório e deverá ser entregue, impreterivelmente, até o dia 21 de junho de 2010 (segunda-feira). A solução é individual e deverá ser arquivada no diretório “~alexnd/TAP/” onde o nome do arquivo terá como prefixo o seu nome-de-usuário no sistema do laboratório e, como extensão, “.p” para indicar que seu conteúdo possui um programa em Prolog. Assim, por exemplo, se o seu nome de usuário no sistema fosse “grs07” então o nome do arquivo seria “grs07.p” (dentro do diretório “~alexnd/TAP/”). Não se esqueça de proteger completamente o arquivo criado, de maneira a permitir a leitura do mesmo apenas por você! Isso pode ser feito aplicando `chmod og-rwx grs07.p` antes de efetuar a cópia com a preservação das permissões (`cp -p grs07.p ~alexnd/TAP/`). Não se preocupe com as permissões do professor que irá corrigir o trabalho. A correção dos trabalhos será parcialmente automatizada, sendo assim, é importante que todos os arquivos com as soluções individuais estejam no diretório citado acima, dentro do prazo estipulado. Não será permitida a entrega do arquivo por e-mail.

## Enunciado:

O problema clássico de atendimento integrado das chamadas a um grande número de elevadores pode ser muito complexo, sendo assim, são adotadas aqui algumas simplificações. As principais delas estão listadas abaixo:

- só há 1 (um) elevador em operação;
- a cabine do elevador tem capacidade ilimitada (nunca fica cheia);
- cada andar possui apenas um botão indicador de chamada do elevador;
- cada pessoa faz uma, e somente uma, chamada quando está fora da cabine do elevador;
- cada pessoa faz uma, e somente uma, requisição quando está dentro da cabine do elevador;
- o prédio onde o elevador se encontra possui a quantidade  $N + 1$  de andares, numerados de 0 (térreo) até  $N$ ;
- se for possível, o planejamento do atendimento das chamadas e/ou requisições deve ser feito com base em apenas uma subida da cabine do elevador até o andar mais alto para o qual houve chamada e/ou requisição, e uma descida até o andar mais baixo para o qual houve requisição;
- é assumido o funcionamento totalmente sincronizado do atendimento das chamadas e requisições (isso significa que, uma vez iniciado o atendimento de um conjunto de chamadas, nenhuma nova chamada é considerada até que as pendentes e as requisições que delas derivaram sejam atendidas por completo).

Fazer um programa orientado a objetos em linguagem Flavors do ambiente Poplog, contendo apenas o código das classes, o qual permite a criação de instâncias capazes de responder às mensagens exemplificadas abaixo:

```
: vars e1 p1 p2 p3 p4 f1 f2 f3;
: make_instance([ elevador nome e1 altura 8 andar 0 ]) -> e1;
: make_instance([ pessoa nome p1 andar 3 meu_elevador e1 ]) -> p1;
: make_instance([ pessoa nome p2 andar 3 meu_elevador e1 ]) -> p2;
: make_instance([ pessoa nome p3 andar 6 meu_elevador e1 ]) -> p3;
: make_instance([ pessoa nome p4 andar 7 meu_elevador e1 ]) -> p4;
: make_instance([ funcionario nome f1 andar 5 meu_elevador e1 ]) -> f1;
: make_instance([ funcionario nome f2 andar 4 meu_elevador e1 ]) -> f2;
: make_instance([ funcionario nome f3 andar 0 meu_elevador e1 ]) -> f3;
: p1 <- saia_do_predio;
Eu, p1, chamei o elevador.
: p2 <- saia_do_predio;
Eu, p2, chamei o elevador.
: p3 <- saia_do_predio;
Eu, p3, chamei o elevador.
: p4 <- vai_para_casa;
Eu, p4, chamei o elevador.
: f1 <- limpe_o_andar(7);
Eu, f1, chamei o elevador.
: f2 <- saia_do_predio;
```

```

Eu, f2, chamei o elevador.
: f3 <- limpe_o_andar(6);
Eu, f3, chamei o elevador.
: e1 <- atenda_chamadas_e_requisicoes;
Eu, e1, dei inicio ao atendimento de chamadas e requisicoes.
Eu, f3, entrei na cabine.
Eu, f3, apertei o botao para ir ao andar 6.
Eu, p4, entrei na cabine.
Eu, p4, apertei o botao para ir ao andar 7.
Eu, e1, fechei a porta.
Eu, e1, deixei o andar 0 e subi.
Eu, e1, passei direto pelo andar 1 na subida.
Eu, e1, passei direto pelo andar 2 na subida.
Eu, e1, passei direto pelo andar 3 na subida.
Eu, e1, parei no andar 4 e abri a porta.
Eu, f2, entrei na cabine.
Eu, f2, apertei o botao para ir ao andar 0.
Eu, e1, fechei a porta.
Eu, e1, deixei o andar 4 e subi.
Eu, e1, parei no andar 5 e abri a porta.
Eu, f1, entrei na cabine.
Eu, f1, apertei o botao para ir ao andar 7.
Eu, e1, fechei a porta.
Eu, e1, deixei o andar 5 e subi.
Eu, e1, parei no andar 6 e abri a porta.
Eu, f3, sai da cabine.
Eu, p3, entrei na cabine.
Eu, p3, apertei o botao para ir ao andar 0.
Eu, e1, fechei a porta.
Eu, e1, deixei o andar 6 e subi.
Eu, e1, parei no andar 7 e abri a porta.
Eu, f1, sai da cabine.
Eu, p4, sai da cabine.
Eu, e1, fechei a porta.
Eu, e1, deixei o andar 7 e descii.
Eu, e1, passei direto pelo andar 6 na descida.
Eu, e1, passei direto pelo andar 5 na descida.
Eu, e1, passei direto pelo andar 4 na descida.
Eu, e1, parei no andar 3 e abri a porta.
Eu, p2, entrei na cabine.
Eu, p2, apertei o botao para ir ao andar 0.
Eu, p1, entrei na cabine.
Eu, p1, apertei o botao para ir ao andar 0.
Eu, e1, fechei a porta.
Eu, e1, deixei o andar 3 e descii.
Eu, e1, passei direto pelo andar 2 na descida.
Eu, e1, passei direto pelo andar 1 na descida.
Eu, e1, parei no andar 0 e abri a porta.
Eu, p1, sai da cabine.
Eu, p2, sai da cabine.
Eu, p3, sai da cabine.
Eu, f2, sai da cabine.
Eu, e1, terminei o atendimento de chamadas e requisicoes.

```

As seguintes especificações adicionais se aplicam ao problema e devem ser consideradas integralmente:

- o tempo que a cabine do elevador fica parada com a porta aberta em cada andar é  $T = T_s + T_e$  segundos, onde  $T_s$  é o número de pessoas que saem da cabine e  $T_e$  é o número de pessoas que entram (o efeito gráfico deve ser traduzido na transformação do retângulo correspondente ao andar para vídeo reverso por um total de  $T$  segundos, seguido de sua restauração para vídeo normal – veja as funções gráficas auxiliares abaixo);
- o tempo de passagem da cabine do elevador por cada andar (sem parar nele) é de aproximadamente 1/3 de segundo (o efeito gráfico deve ser traduzido na transformação do retângulo correspondente ao andar para vídeo reverso por um total

de 1/3 de segundo, seguido de sua restauração para vídeo normal – veja as funções gráficas auxiliares abaixo);

- as duas únicas categorias de pessoas que circulam pelo prédio são os moradores comuns do prédio e os funcionários;
- em qualquer instante, cada morador só tem as opções de entrar no prédio, indo direto para o andar de seu apartamento (identificado por meio da variável de instância “andar”), ou de sair do prédio (nunca de mudar de andar);
- em qualquer instante, cada funcionário só tem as opções de limpar um andar específico ou de sair do prédio (ele pode mudar de andar e a variável de instância “andar” significa a sua localização inicial, o que é diferente do significado dela para os moradores);
- as diversas chamadas e requisições feitas ao elevador são identificadas com o “nome” de cada pessoa (o elevador utiliza um sistema de código com identificação pessoal);
- em qualquer instante, cada funcionário requisitado ao trabalho deverá ficar responsável pela limpeza de um andar diferente;
- a tarefa de limpar um andar, além de ser restrita aos funcionários, será cumprida meramente por meio da mudança de andar por parte do funcionário responsável (ao chegar no andar da limpeza, o funcionário lá ficará permanentemente);
- a única otimização referente à solução deste problema (escoamento ordenado de entrada e saída) faz com que o elevador dê preferência para subir pois tenta atender às requisições de quem chega da rua assim como às chamadas de quem quer sair (ele tenta subir o máximo possível para conduzir os que entram no prédio na medida em que apanha as pessoas nos andares inferiores no sentido de saída);
- isso faz com que chamadas de moradores provoquem a parada do elevador apenas no sentido de descida;
- todavia, chamadas de funcionários sempre provocam a parada do elevador no sentido de subida;
- se o elevador parar na subida, por qualquer razão, todos que estão no andar são atendidos.

Há também um painel gráfico que deve simular o movimento do elevador a partir da coloração de  $N + 1$  retângulos dispostos verticalmente. Para desenvolver os efeitos gráficos no contexto das classes acima especificadas, o arquivo de biblioteca “elevadorgraf.p” deve ser carregado. O referido arquivo está disponível em [www.inf.ufpr.br/alex/d/tap/elevadorgraf.p](http://www.inf.ufpr.br/alex/d/tap/elevadorgraf.p) para cópia. As 4 (quatro) primitivas gráficas do arquivo que serão necessárias são as seguintes:

1 - `redesenha_arvore(<ESTRUTURA_DE_ARVORE>);`

Este é um Procedimento que recebe uma árvore n-ária como argumento e tem a responsabilidade de apagar o conteúdo da superfície gráfica de visualização e de redesenhar uma hierarquia de nomes de acordo com a organização interna de subordinação dos referidos nomes na estrutura de dados.

2 - `monta_lista_coord(<ESTRUTURA_DE_ARVORE>)`

Esta é uma Função que recebe uma árvore n-ária como argumento e devolve uma lista de listas contendo dois elementos: uma “word” (< *NOME\_NODO* >) e uma lista com 2 (duas) coordenadas – (X1,Y1) e (X2,Y2) – as quais representam, respectivamente, os vértices superior esquerdo e inferior direito do retângulo gráfico do nodo da árvore que tem o nome de < *NOME\_NODO* >. Veja o exemplo abaixo:

```
: monta_lista_coord([andar_8 [andar_7 [andar_6 [andar_5 [andar_4 [andar_3 [andar_2 [andar_1 [andar_0]]]]]]]]) ==>
** [[andar_7 [1 5 11 7]]
    [andar_8 [1 1 11 3]]
    [andar_5 [1 13 11 15]]
    [andar_3 [1 21 11 23]]
    [andar_2 [1 25 11 27]]
    [andar_0 [1 33 11 35]]
    [andar_6 [1 9 11 11]]
    [andar_4 [1 17 11 19]]
    [andar_1 [1 29 11 31]]]
```

3 - `video_reverso(<NOME_NODO>, X1, Y1, X2, Y2);`

Este é um Procedimento que recebe, como argumento, 5 elementos: uma “word” (< *NOME\_NODO* >) e quatro valores inteiros representando 2 (duas) coordenadas – (X1,Y1) e (X2,Y2) – as quais representam, respectivamente, os vértices superior esquerdo e inferior direito do retângulo gráfico do nodo da árvore que tem o nome de < *NOME\_NODO* >. Ele produz apenas como efeito colateral a transformação de cores de frente e fundo em video-reverso.

4 - `video_normal(<NOME_NODO>, X1, Y1, X2, Y2);`

Este é um Procedimento idêntico ao do item 3 (acima), porém com o efeito colateral de alterar as cores de frente e de fundo em video-normal.

Recomenda-se assistir à demonstração diretamente com o professor da disciplina para entender os comportamentos das instâncias por meio de um programa-exemplo implementado com entradas e saídas textuais e gráficas. Finalmente, para executar o arquivo da biblioteca gráfica, basta abrir uma “shell” em qualquer máquina com o comando “xterm” do Linux e digitar a seguinte sequência de comandos:

```
dalmore> source ~alex/d/spop
Sussex Poplog Version 15.53
dalmore> pop11 %x
;;; Warning: ...
...
Sussex Poplog (Version 15.53 Sat Nov 6 23:15:43 GMT 1999)
Copyright (c) 1982-1999 University of Sussex. All rights reserved.
...

Setpop
: uses flavours;
: load elevadorgraf.p;
  <<< J A N E L A   G R A F I C A   S U R G E >>>
  <<< EXPERIMENTE DESENHAR UMA ARVORE >>>
: redesenha_arvore([andar_8 [andar_7 [andar_6 [andar_5 [andar_4 [andar_3 [andar_2 [andar_1 [andar_0]]]]]]]]]);
  <<< EXPERIMENTE CLICAR NOS NODOS E FAZE-LOS MUDAR DE COR >>>
```

#### Dica:

Para facilitar a codificação da sequência de envio das mensagens, pode ser interessante organizar seu código para contar com dois métodos, cujos nomes seriam `entre_na_cabine` e `saia_da_cabine`, os quais se aplicam às pessoas do ambinete (moradores e funcionários).

## B O M T R A B A L H O !

No final de tudo, você chegará à conclusão que, em alguns casos, o elevador vai precisar de mais do que apenas uma subida e uma descida para atender às chamadas e requisições. Sendo assim, tente também os seguintes eventos para ver o que acontece com sua solução:

```
: vars e1 p1 p2 p3 p4 f1 ;
: make_instance([ elevador nome e1 altura 8 andar 5 ]) -> e1;
: make_instance([ pessoa nome p1 andar 3 meu_elevador e1 ]) -> p1;
: make_instance([ pessoa nome p2 andar 3 meu_elevador e1 ]) -> p2;
: make_instance([ pessoa nome p3 andar 3 meu_elevador e1 ]) -> p3;
: make_instance([ pessoa nome p4 andar 7 meu_elevador e1 ]) -> p4;
: make_instance([ funcionario nome f1 andar 4 meu_elevador e1 ]) -> f1;
: p1 <- saia_do_predio;
: p2 <- saia_do_predio;
: p3 <- saia_do_predio;
: p4 <- vai_para_casa;
: f1 <- limpe_o_andar(7);
: e1 <- atenda_chamadas_e_requisicoes;
```